EECE.3220: Data Structures

Spring 2019

Exam 2 Solution

1. (20 points) ***Composition***

This section uses the classes `Qscore` *and* `Tscore` *defined on the extra sheet. Given those class and selected function definitions, you must write the remaining functions. Your implementations should be written so the test cases shown on the extra sheet produce the proper output.*

A couple of hints:

- *Each individual question score should be printed using* `Qscore::display()`*, so write your* `Tscore::display()` *definition accordingly.*
- *Read the* `Qscore::fraction()` *definition carefully—the return value is always between 0.0 and 1.0, so you'll have to change that value slightly to get the correct output.*
- *Don't worry about output manipulators and formatting—the test cases were generated using default output settings.*
- *Don't worry if your solution doesn't fill the space available—I just wanted to make sure everyone had enough space to write their solutions without squeezing too many functions onto one page.*

a. (12 points) `void Qscore::display(int i, ostream &out)`*: Print content of* `Qscore` *object to output stream* `out`*. Argument* `i` *represents question number.*

```
void Qscore::display(int i, ostream &out) {
   out << 'Q' << i << ": "
      << pts << '/' << max
      << " (" << fraction() * 100
      << "%)\n";
}
```

b. (8 points) `Tscore::Tscore(int q1s, int q1m, int q2s, int q2m, int q3s, int q3m)`*: Constructor for* `Tscore` *objects. Two notes:*

- *I think argument names make it relatively obvious which data members they should be assigned to, but ask for clarification if necessary.*
- *I've left blank space so you can put your curly braces { } where you want.*

```
Tscore::Tscore(int q1s, int q1m, int q2s, int q2m, int q3s, int q3m) :
      Q1(q1s, q1m), Q2(q2s, q2m), Q3(q3s, q3m) {}
```

c. (20 points) `void Tscore::display(ostream &out)`*: Print content of* `Tscore` *object to output stream* `out`.

```
void Tscore::display(ostream &out) {
   int m1, m2, m3, total;
   Q1.display(1, out);
   Q2.display(2, out);
   Q3.display(3, out);
   m1 = Q1.getMax();
   m2 = Q2.getMax();
   m3 = Q3.getMax();
   total = Q1.fraction() * m1 + Q2.fraction() * m2 +
           Q3.fraction() * m3;

   out << "TOTAL: " << total << '/' << m1 + m2 + m3
      << " (" << total * 100.0 / (m1 + m2 + m3)
      << "%)\n";
}
```

2. (26 points) ***Dynamic allocation***

a. (20 points) Complete the function with the following prototype:

```
int *reallocate(int *arr1, unsigned size1, unsigned size2);
```

Given a pointer `arr1` to a dynamically allocated integer array of size `size1`, allocate a new array of size `size2`, copy data from the old array to the new one, delete the old array, and return the address of the new array.

When copying data, if the new array is larger, copy all data from the old array (leaving the rest of the new array uninitialized), but if the new array is smaller, only copy as much data as the new array can hold.

```
int *reallocate(int *arr1, unsigned size1, unsigned size2) {
   unsigned i;       // Loop index
   int *newArr;      // Pointer to newly allocated array
   int nCopies;      // # values to copy from old to new array

   // Allocate new array
   newArr = new int[size2];

   // Determine number of values to copy from old to new array
   if (size2 >= size1)
      nCopies = size1;
   else
      nCopies = size2;

   // Copy data from old to new array

   for (i = 0; i < nCopies; i++) {
      newArr[i] = arr1[i];
   }

   // Delete old array, then return address of new array
   delete [] arr1;
   return newArr;
}
```

2 (continued)

b. (6 points) You are given the constructor below for a class `C2b`*:*

```
C2b::C2b(unsigned x, unsigned y, unsigned z) : m1(x) {
   m2 = new double[y];
   m3 = new double(z);
}
```

Which of the choices below show a valid destructor for this class, based on the constructor above?
This question has only one correct answer.

i.
```
C2b::~C2b() {
   delete m1;
   delete m2;
   delete m3;
}
```

ii.
```
C2b::~C2b() {
   delete m2;
   delete m3;
}
```

iii.
```
C2b::~C2b() {
   delete m2;
   delete [] m3;
}
```

iv.
```
C2b::~C2b() {
   delete [] m2;
   delete m3;
}
```

3. *(34 points)* ***Stacks***

a. *(16 points) For the short program shown below, list the output exactly as it will appear on the screen. Be sure to clearly indicate spaces between characters when necessary.*

You may use the available space to show your work as well as the output; just be sure to clearly mark where you show the output so that I can easily recognize your final answer.

This question assumes the use of the `Stack` *implementation covered in lecture; the class definition is on the extra sheet. Function definitions aren't shown because it's assumed you understand what the different* `Stack` *functions do.*

```
int main() {
   int arr1[7] = { 8, 6, 7, 5, 3, 0, 9 };
   double arr2[10] = { 0, 1.1, 2.2, 3.3, 4.4,
                       5.5, 6.6, 7.7, 8.8, 9.9 };
   int i, j;
   Stack S1;

   for (i = 0; i < 7; i++) {
      j = arr1[i];                 Pushes 8.8, 6.6, 7.7, 5.5, 3.3, 0,
      S1.push(arr2[j]);               and 9.9 in that order
   }

   while (!S1.empty()) {
      cout << S1.top() << endl;    Prints values in stack in reverse
      S1.pop();                       of order they were pushed
   }
   return 0;
}
```

OUTPUT:

```
9.9
0
3.3
5.5
7.7
6.6
8.8
```

3 (continued)

b. (6 points) Which of the declarations below creates the `Stack` *object with the greatest capacity?* ***This question assumes the use of the*** `Stack` ***class definition on the extra sheet and has exactly ONE correct answer.***

i. **`Stack S1;`** **`(uses default argument, maxSize = 1024)`**

ii. `Stack S2(10);`

iii. `Stack S3(100);`

iv. All `Stack` objects have the same capacity

c. (6 points) Which of the following statements correctly describes the details of a linked stack? Please circle ***all*** *correct choices—****this question has at least one correct answer, but may have more than one correct answer!***

i. An empty linked stack contains no dynamically allocated data, just a single null pointer.

ii. The pointer in a linked stack object should always point to the first node created, and each node, *N*, should point to the node created after node *N*.

iii. The pop operation in a linked stack requires an extra pointer so you can store the address of the old top node, change the top pointer to point to the second node, and then use the extra pointer to delete the old top node.

iv. The destructor in a linked stack only needs to delete the top node of the stack, regardless of how many nodes are in the stack.

d. (6 points) Which of the following statements accurately reflect your opinion(s)? Circle all that apply (but please don't waste too much time on this "question")!

i. "I think the most recent assignments (Programs 2 and 3) are relatively easy."

ii. "I think the most recent assignments are too difficult."

iii. "I think I've answered enough questions for one day."

iv. "Is the semester over yet?"

All answers are "correct."

EXTRA CREDIT *(10 points)*

Show the output of the program below exactly as it appears on screen. You may use the next page to show your work and final output if this page does not have enough space.

Assume the use of the `Stack` *defined on the extra sheet (so every value on the stack is stored as a double), and assume the user input is below (assume a newline follows the final* `'-'`*):*

```
9 7 1 3 + - / 4 * 2 5 1 + + -
```

```
int main() {
   Stack S1;
   char c = 0;
   double val, val2, result;
   cout << "Enter expression: ";
   do {
      cin >> c;
      if (isdigit(c)) {               // True if c represents a number
         S1.push(c - '0');
         cout << "Pushed " << S1.top() << endl;
      }

      else {
         val = S1.top();
         S1.pop();
         val2 = S1.top();
         S1.pop();
         if (c == '+')
            result = val2 + val;
         else if (c == '-')
            result = val2 - val;
         else if (c == '*')
            result = val2 * val;
         else if (c == '/')
            result = val2 / val;
         S1.push(result);
         cout << "New TOS = " << S1.top() << endl;
      }
   } while (cin.get() != '\n');
   cout << "Final result: " << S1.top();

   return 0;
}
```

Solution: This program does the following:

- Reads a non-space character at the start of each iteration
- If that character represents a digit, that digit is pushed on the stack.
  - `c – '0'` converts a numeric character to the integer it represents. `'0' – '0' = 0`, `'1' – '0' = 1`, and so on.
- If that character represents an operation, two values are popped off the stack, operated on, and the result is pushed back on the stack.
  - The topmost value is on the right of the operand, so the input `7 10 –` would result in the operation: `10 – 7 = 3`
- The cin.get() call in the loop condition reads each whitespace character that follows a number or operator, causing the loop to stop when the newline at the end of the line is read.

The table below shows what each input character is, how the stack is changed, and what the output is for each loop iteration. The top of the stack is always shown on the left in the "Stack state" column.

Input	Operation	Stack state	Output
9	Push 9	9	`Pushed 9`
7	Push 7	7 9	`Pushed 7`
1	Push 1	1 7 9	`Pushed 1`
3	Push 3	3 1 7 9	`Pushed 3`
+	Add top 2 stack values & push result: 1 + 3 = 4	4 7 9	`New TOS = 4`
-	Subtract top 2 stack values & push result 7 – 4 = 3	3 9	`New TOS = 3`
/	Divide top 2 stack values & push result: 9 / 3 = 3	3	`New TOS = 3`
4	Push 4	4 3	`Pushed 4`
*	Multiply top 2 stack values & push result: 3 * 4 = 12	12	`New TOS = 12`
2	Push 2	2 12	`Pushed 2`
5	Push 5	5 2 12	`Pushed 5`
1	Push 1	1 5 2 12	`Pushed 1`
+	Add top 2 stack values & push result: 5 + 1 = 6	6 2 12	`New TOS = 6`
+	Add top 2 stack values & push result: 2 + 6 = 8	8 12	`New TOS = 8`
-	Subtract top 2 stack values & push result: 12 – 8 = 4	4	`New TOS = 4`
			`Final result: 4`